UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN
INFRAESTRUCTURA DE COMUNICACIONES
Profesor: Harold Enrique Castro Barrera
hcastro@uniandes.edu.co
Asist. De laboratorio: Jonatan Legro Pastrana
j.legro@uniandes.edu.co

LABORATORIO 5 – TRANSMISIÓN DE DATOS SOBRE UDP

1. OBJETIVO (S)

UDP (User Datagram Protocol) es un protocolo de transporte utilizado en Internet como alternativa a TCP cuando no se requiere confiabilidad. El objetivo del presente laboratorio se centra en implementar sockets UDP para establecer comunicación entre un servidor y un cliente.

Al finalizar la práctica, el estudiante estará en capacidad de:

- Desplegar una máquina virtual en Amazon Web Services o Microsoft Azure u otra nube.
- Desarrollar una aplicación cliente para enviar/recibir soportada en el uso de sockets UDP.
- Desarrollar una aplicación servidor para enviar/recibir datos soportada en el uso de sockets UDP.
- Desarrollar una aplicación cliente para enviar/recibir soportada en Bittorrent.
- Desarrollar una aplicación servidor para enviar/recibir datos soportada en Bittorrent.
- Comparar el desempeño de las aplicaciones cliente/servidor desarrolladas en TCP, UDP y Bittorrent en términos de la entrega confiable de información y el tiempo de transferencia.

2. LECTURAS PREVIAS

Lecturas recomendadas:

- Generalidades de los protocolos UDP [1]
- Implementación de sockets en Java [2]
- Implementación de sockets en Python [3, 4]
- Amazon Web Services [5, 6]

3. INFORMACIÓN BÁSICA

Para motivar el uso de UDP en la práctica de laboratorio, suponga que está interesado en diseñar un protocolo de transporte sin overhead. ¿Cómo podría desarrollar este protocolo? Primero, considere usar un protocolo de transporte vacío. En el emisor, puede tomar los mensajes de la aplicación y pasarlos directamente a la capa de internet; en el receptor, puede tomar los mensajes que llegan de la capa de red y pasarlos directamente a la aplicación. Sin embargo, a nivel de transporte se requiere

proporcionar como mínimo un servicio de multiplexación / demultiplexación para pasar datos entre la capa de red y la aplicación correcta.

UDP hace tan poco como un protocolo de transporte puede. Aparte de la función de multiplexación / demultiplexación, realiza comprobación de errores para los encabezados en capa de transporte. De hecho, si un desarrollador selecciona UDP en lugar de TCP, entonces la aplicación está hablando casi directamente con la capa de internet.

UDP toma los datos de la capa de aplicación, agrega los campos de número de puerto de origen y destino para el servicio de multiplexación / demultiplexación, agrega otros dos campos de menor importancia y pasa el "segmento o datragrama" resultante a la capa de red. La capa de red encapsula el datagrama en un paquete IP y luego realiza un intento de mejor esfuerzo para entregar el datagrama al receptor.

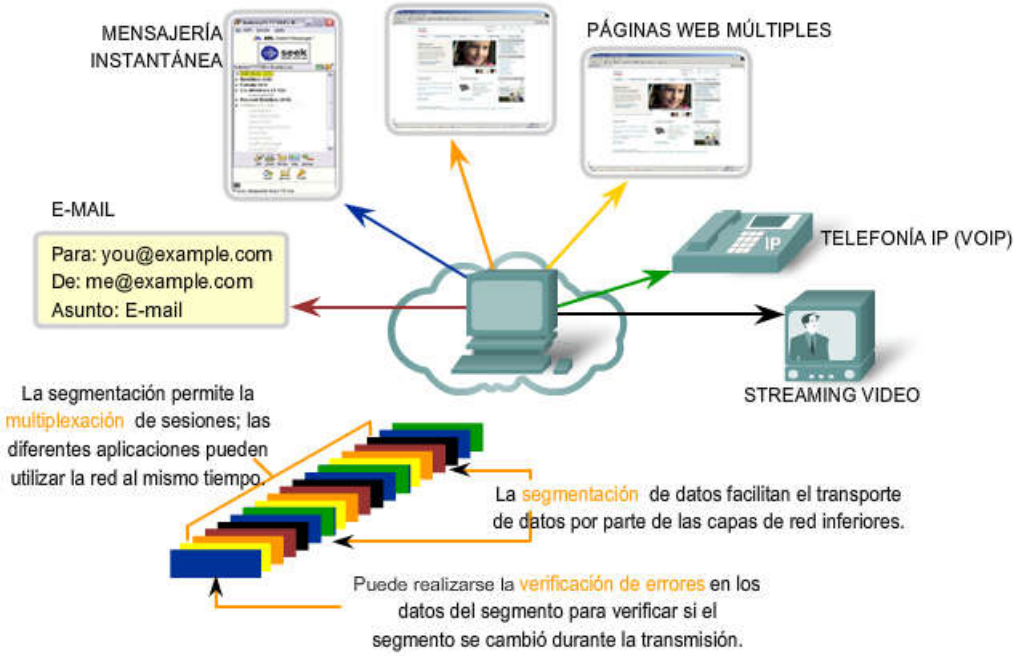

Ilustración 1. Servicios de la capa de transporte

UDP se utiliza comúnmente en aplicaciones multimedia, como la telefonía IP, streaming de audio y video, dado que todas estas aplicaciones pueden tolerar pequeñas pérdidas de información, por lo que la transferencia de datos confiable no es absolutamente crítica para el éxito de la aplicación. Finalmente, debido a que TCP no se puede emplear con multidifusión, las aplicaciones de multidifusión se ejecutan a través de UDP.

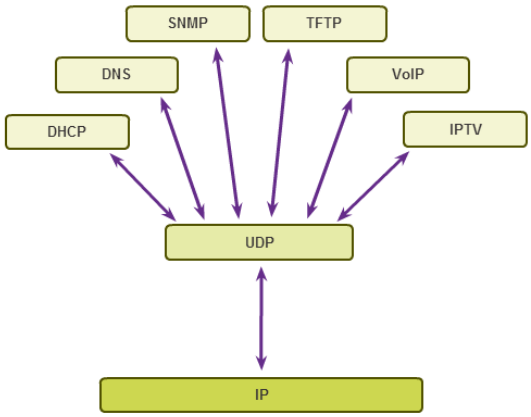

Ilustración 2. Aplicaciones que utilizan UDP

En esta práctica se desarrollará aplicaciones orientadas a la red soportada en UDP, un cliente y un servidor. El laboratorio se centra en el intercambio de mensajes y archivos entre cliente y servidor, por medio de UDP, y el despliegue del servidor en Amazon Web Services.

La práctica finaliza con una evaluación del protocolo de comunicación implementado, y una comparación con otros mecanismos de entrega confiable de archivos, como lo es TCP y el servicio de capa de aplicación Bittorent.

4. PROCEDIMIENTO

El objetivo inicial es desarrollar una aplicación que permita enviar un conjunto ordenado de datos, entre un cliente y un servidor UDP. El ejercicio está compuesto de dos aplicaciones que se comunican entre sí usando sockets UDP e implementando persistencia sobre archivos de texto.

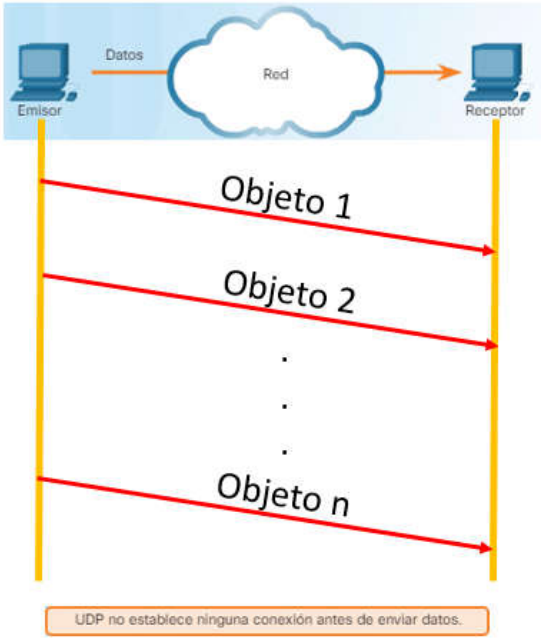

Ilustración 3. Transmisión UDP.

4.1. Requerimientos cliente UDP

Desarrollar una aplicación cliente en el lenguaje de programación de su preferencia (e.g., Java, Python, etc). Este deberá cumplir con los siguientes requerimientos:

1. Enviar un conjunto de objetos en forma ordenada desde un cliente UDP a un servidor UDP. El objeto debe contener un valor numérico entero que identifica la posición del objeto y una marca de tiempo asociada al momento en el que es enviado el objeto.

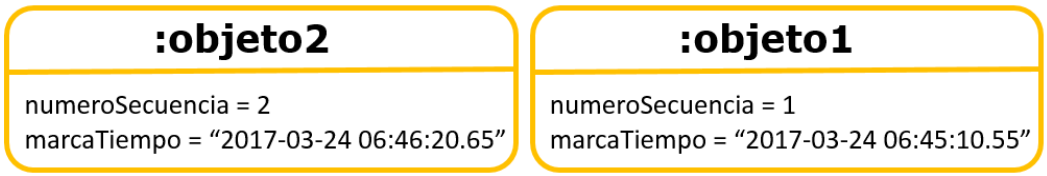

Ilustración 4. Ejemplo objetos a enviar vía UDP.

2. Desarrollar una interfaz de usuario simple (puede ser de consola y no grafica) que permita ajustar los siguientes parámetros: dirección IP del servidor, puerto en el que escucha y recibe el servidor, número de objetos a generar y enviar.

4.2. Requerimientos servidor UDP

Desarrollar un servidor en el lenguaje de programación de su preferencia (e.g., Java, Python, etc). Este deberá cumplir con los siguientes requerimientos:

1. Recibir un conjunto de objetos vía UDP.

2. La información enviada por un cliente debe ser almacenados en un archivo, el nombre del archivo debe ser único por cada cliente. La estructura del archivo es la siguiente: un dato por línea, debe contener el identificador numérico del objeto y el tiempo tomado en la transmisión (diferencia entre la marca de tiempo en el que se recibió el dato y la marca de tiempo en el que fue enviado): Ejemplo:

   1: 53 ms
   2: 52 ms
   3: 52 ms
   4: 53 ms
   7: 55 ms

3. Como se mencionó la aplicación no requiere interfaz gráfica, pero si debe permitir cambiar el número de puerto vía línea de comandos al momento de ser ejecutada. Ejemplo:

   python servidorUDP.py 5000

   java –jar servidorUDP.jar 5000

4. La aplicación debe permitir identificar si uno o más objetos se perdieron en tránsito (no es necesario identificar cuales).

5. La aplicación debe presentar cifras como: número de objetos recibidos, numero de objetos faltantes, tiempo promedio de envió de objetos. Estos datos pueden ser presentados en la aplicación o ser almacenados en el archivo generado para cada cliente.

6. El servidor debe poder atender el envío de varios clientes.

7. El servidor y el cliente deben poder funcionar en máquinas diferentes.

4.3. Despliegue

1. El despliegue de la aplicación servidor debe realizarse sobre una instancia de Amazon Web Services. "Amazon Elastic Compute Cloud (EC2) es el servicio de Amazon Web Services que se utiliza para crear y ejecutar máquinas virtuales en la nube (las denominamos "instancias)" [5].

2. Cree una instancia Ubuntu/Linux de Amazon EC2. Si nunca ha realizado este procedimiento siga la referencia [6], esta presenta un tutorial desarrollado por Amazon para realizar este procedimiento.

3. Recuerde si no tiene una cuenta de Amazon Web Services, puede crear de forma gratuita una cuenta nueva y acceder al servicio. Amazon Web Services ofrece una capa gratuita que le facilita una instancia Linux T2 Micro durante 750 horas al mes.

4. **Advertencia**: Si va a utilizar servicios como Git Hub o Bitbucket y su repositorio es público, **NO** almacene en estos la llave de su instancia por seguridad.

5. Recuerde habilitar los puertos UDP de su preferencia para la aplicación servidor cuando este modificando la configuración de la instancia.

Ilustración 5. Configuración de puertos para una instancia de Amazon Web Services

6. Suba su ejecutable del servidor UDP a la instancia Linux de Amazon Web Services, y ejecute ésta en el puerto de su preferencia.

7. Despliegue de forma local la aplicación cliente y realice pruebas de la comunicación entre cliente y servidor. Valide que comunicación entre ambas aplicaciones opere de forma correcta.

8. **netstat** es una utilidad de estadísticas de red que se encuentra disponible tanto en sistemas operativos Windows como en sistemas Unix/Linux. El paso de parámetros opcionales con el comando cambiará la información de resultado. **netstat** muestra conexiones de red entrantes y salientes (TCP y UDP), información de tabla de enrutamiento del equipo host y estadísticas de la interfaz.

9. Durante la ejecución de las pruebas propuestas utilice el comando netstat para identificar las conexiones UDP activas. **Presente en una tabla la información obtenida**. **¿todas las aplicaciones cliente utilizan el mismo puerto local? Explique**.

10. Para la prueba solicitada a continuación ejecute Wireshark en el cliente y active la captura de paquetes.

11. Realice pruebas enviando 1.000, 10.000 y 100.000 objetos desde el cliente.

12. Repita las pruebas anteriores enviando la misma cantidad de datos desde 3 clientes diferentes.

13. **Presente los resultados obtenidos de las pruebas anteriores en una tabla**. Desde Wireshark verifique la información UDP capturada.

4.4. Intercambio de archivos en UDP

1. Modifique la aplicación cliente y la aplicación servidor, de tal forma que se permita el envío de archivos desde el servidor al cliente. Los archivos enviados pueden ser de cualquier tipo y tamaño.

2. Tanto la aplicación cliente como la aplicación servidor deben permitir ajustar el tamaño del buffer de envío y el tamaño del buffer de recepción para las diferentes pruebas a realizar.

3. La aplicación servidor debe calcular un hash del archivo solicitado para transferencia, cualquier algoritmo es válido del archivo antes del proceso de envío (e.g., MD5, SHA, etc).

4. La aplicación cliente debe calcular el hash del archivo (utilice el mismo algoritmo del punto anterior) luego de que este es recibido y almacenado en disco.

5. Compare el código de hash en el cliente con el código de hash en el servidor para validar que el archivo llego completo y correcto.

6. Las aplicaciones deben permitir modificar el tamaño del buffer de envío y recepción de datos.

7. Las aplicaciones deben permitir medir el tiempo de transferencia de un archivo en segundos. Este tiempo debe calcularse desde el momento en que se envía el primer paquete en el servidor hasta el momento en el que se recibe el último paquete en el cliente. Al final de cada transferencia la aplicación debe reportar si el archivo está completo y correcto y el tiempo total de transferencia.

8. Seleccione un fichero con un tamaño inferior a 2MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

9. Seleccione un fichero con un tamaño superior a 5MiB e inferior a 10MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

10. Seleccione un fichero con un tamaño superior a 50MiB e inferior a 100MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

11. Presente los resultados de las pruebas realizadas.

12. Para la prueba solicitada a continuación ejecute Wireshark en el cliente y active la captura de paquetes.

13. **Repita las pruebas anteriores modificando el tamaño del buffer de envío con los siguientes valores: mínimo tamaño permitido del buffer, máximo tamaño permitido del buffer, un valor intermedio. Realice la misma prueba varias veces (al menos en 3 ocasiones) para que analice el comportamiento de las aplicaciones. Valide en cada prueba si el archivo fue recibido correctamente.**

14. **Verifique la captura realizada en Wireshark para cada uno de los escenarios y cada una de las pruebas, identifique el tamaño de los datagramas UDP y la cantidad de datagramas enviados.**

4.5. Intercambio de archivos en Bittorrent

1. Desarrolle una aplicación cliente y una aplicación servidor, de tal forma que se permita el envío de archivos desde el servidor al cliente soportado en el protocolo Bittorrent para el intercambio de archivos en entornos P2P. Los archivos enviados pueden ser de cualquier tipo y tamaño.

2. Las aplicaciones deben permitir medir el tiempo de transferencia de un archivo en segundos. Este tiempo debe calcularse desde el momento en que se envía el primer paquete en el servidor hasta el momento en el que se recibe el último paquete en el cliente. Al final de cada transferencia la aplicación debe reportar si el archivo está completo y correcto y el tiempo total de transferencia.

3. Despliegue la aplicación servidor en la instancia de AWS utilizada en la sección 4.3 y la aplicación cliente en los equipos del laboratorio.

4. Seleccione un fichero con un tamaño inferior a 2MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

5. Seleccione un fichero con un tamaño superior a 5MiB e inferior a 10MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

6. Seleccione un fichero con un tamaño superior a 50MiB e inferior a 100MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

7. Presente los resultados de las pruebas realizadas.

8. Para la prueba solicitada a continuación ejecute Wireshark en el cliente y active la captura de paquetes.

9. **Verifique la captura realizada en Wireshark para cada uno de los escenarios y cada una de las pruebas, identifique el tamaño y la cantidad de paquetes intercambiados entre cliente y servidor.**

4.5. Intercambio de archivos en TCP

1. Realice los ajustes necesarios para utilizar aplicación cliente y la aplicación servidor desarrolladas en la práctica de laboratorio anterior para garantizar el envió de archivos desde el servidor al cliente soportados en el protocolo de transporte TCP. Los archivos enviados pueden ser de cualquier tipo y tamaño.

2. Las aplicaciones deben permitir medir el tiempo de transferencia de un archivo en segundos. Este tiempo debe calcularse desde el momento en que se envía el primer paquete en el servidor hasta el momento en el que se recibe el último paquete en el cliente. Al final de cada transferencia la aplicación debe reportar si el archivo está completo y correcto y el tiempo total de transferencia.

3. Despliegue la aplicación servidor en la instancia de AWS utilizada en la sección 4.3 y la aplicación cliente en los equipos del laboratorio.

4. Seleccione un fichero con un tamaño inferior a 2MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

5. Seleccione un fichero con un tamaño superior a 5MiB e inferior a 10MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

6. Seleccione un fichero con un tamaño superior a 50MiB e inferior a 100MiB y realice diferentes pruebas de envío. Verifique si en todas las ocasiones el archivo fue recibido correctamente.

7. Presente los resultados de las pruebas realizadas.

8. Para la prueba solicitada a continuación ejecute Wireshark en el cliente y active la captura de paquetes.

9. **Verifique la captura realizada en Wireshark para cada uno de los escenarios y cada una de las pruebas, identifique el tamaño y la cantidad de paquetes intercambiados entre cliente y servidor.**

6. ENTREGABLES

- Informe digital que presente el proceso de solución de los requerimientos efectuados en cada una de las secciones de la práctica, además deben resolver todas las preguntas solicitadas. Presente el análisis de resultados y conclusiones sobre los diferentes escenarios de pruebas realizados, compare los resultados de los 3 protocolos estudiados.

- Responda las siguientes preguntas:
    - ¿Es posible desarrollar aplicaciones UDP que garanticen la entrega confiable de archivos? Que consideraciones deben tenerse en cuenta para garantizar un servicio de entrega confiable utilizando dicho protocolo. Justifique su respuesta.

- Código fuente de las aplicaciones desarrolladas en la práctica, debidamente documentado.

Nota: Deben hacer una sola entrega, en una sola carpeta comprimida con todos archivos. Entrega por Sicua Plus.
Si no poseen cuenta gratuita en AWS pueden trabajar con otra cuenta gratuita de Microsoft Azure, Digital Ocean, Google Cloud.

7. REFERENCIAS

[1] Kurose, James. Ross, Keith. Computer Networking: A Top-Down Approach. 5th edition. Addison-Wesley. Capítulos 2 y 3.

[2] The Java Tutorials. Trail: Custom Networking. http://docs.oracle.com/javase/tutorial/networking/index.html (Revisado el 27 de agosto de 2013).

[3] Rhodes, Brandon. Goerzen, John. Foundations of Python Network Programming. 2nd edition. 2010.

[4]Documentación oficial Python: Socket – Low-level Networking Interface. https://docs.python.org/2/library/socket.html (Revisado el 25 de febrero de 2015).

[5] https://aws.amazon.com/es/

[6] https://aws.amazon.com/es/start-now/

HISTORIAL DE REVISIONES

FECHA	AUTOR	OBSERVACIONES
24/03/2017	**Jesse Padilla Agudelo** pa.jesse10@uniandes.edu.co	Versión inicial del documento.
09/03/2018	**Jesse Padilla Agudelo** pa.jesse10@uniandes.edu.co	Se anexan los requerimientos de Bittorent y TCP.